Introduction

- This was David Penny's research topic.
- Want a (Java) program to help a software company plan new releases of their software (i.e., alpha/beta/sneak peeks, etc.);
- 6 Java files: features.xml, Planets.xml, 546
- XML file contains sized (in coded days) prioritized (hi, med, low) feature requests for various products
- Includes date of requesting product, with how much they want till 3.10.
- Suggest an "optimal" release plan given the available capacity (in coded days).
- Sample output

OOA

- See http://www.aaaweb.org/Models/Overview.html
- Introduction
  - Why are we doing this?
  - What is the current document for?
  - Where did the information come from?
  - General points: change & XML file in this case
- Use Cases
  - What is the biggest problem?
  - How does this particular program solve it?
- Class Diagrams
  - Include information from the requirements statement in UML.
  - (exactly why it's in "requirement implementation")

OOA/OOD/OOP Example

See http://www.aaaweb.org/Models/Overview.html

Release Planning Program
OOD

- Sue ood document
- David's presentation is excellent.
- Package design
  - what rationale for the package breakdown
  - Main driver
    - sequence diagram explaining how (view) the case is executed
  - For each package:
    - collection of class diagrams
      - shows important methods
      - shows important attributes
      - (any association navigability)
      - indicates lower association are implementing
      - indicates that interfaces are being implemented

About Source and Javadoc

- Javadoc is a tool that extracts comments formatted in a certain manner and produces web pages documenting the details of a class design.
  - for example:
    - To display source code, I used a tool called javadoc for pretty-printing Java source to HTML.
  - for example:

Experiments show...

```java
public double plantLeaves(double capacity) {
    double implant = 0.0;
    Feature feature = new Feature();
    feature.addFeature(new Feature());
    Feature feature2 = new Feature();
    Feature feature3 = new Feature();
    Feature feature4 = new Feature();
    Feature feature5 = new Feature();
    Feature feature6 = new Feature();
    feature = feature.addFeature(feature2.addFeature(feature3.addFeature(feature4.addFeature(feature5.addFeature(feature6.addFeature(feature)))));
    return x;
}
```
Implementing Associations

- Decide on navigability
  - The direction in which the navigation can be efficiently multiplied
    - If you have an object of the left class, can you (or OOA) first work all objects of the right class related to the left object?
- Decide on interface for
  - Navigating the links
    - Usually get connected to a side, reverse the "side"
  - Adding new links
  - Deleting links (if necessary)
- Decide on implementation
  - Simple pointer to implement the [0..1] side
    - (if required by navigability)
    - Array, Vector, Map, Linked List to do the [*] side
    - (if required by navigability)